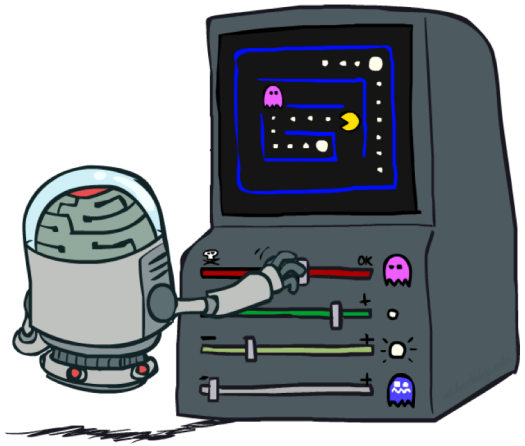
CSE 573: Artificial Intelligence

Hanna Hajishirzi

Reinforcement Learning II



slides adapted from
Dan Klein, Pieter Abbeel ai.berkeley.edu
And Dan Weld, Luke Zettelmoyer

Reinforcement Learning

- Still assume a Markov decision process (MDP):
  - A set of states $s \in S$
  - A set of actions (per state) A
  - A model T(s,a,s')
  - A reward function R(s,a,s')
- Still looking for a policy $\pi(s)$

- New twist: don't know T or R
  - I.e. we don't know which states are good or what the actions do
  - Must actually try actions and states out to learn
- Big Idea: Compute all averages over T using sample outcomes

The Story So Far: MDPs and RL

Known MDP: Offline Solution

Goal	Technique
Compute V*, Q*, π*	Value / policy iteration
Evaluate a fixed policy π	Policy evaluation

Unknown MDP: Model-Based

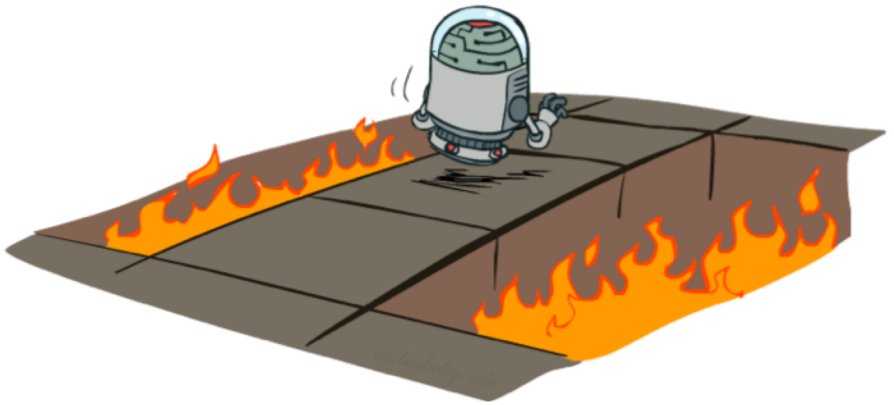
Goal	Technique
Compute V*, Q*, π*	VI/PI on approx. MDP
Evaluate a fixed policy π	PE on approx. MDP

Unknown MDP: Model-Free

Goal	Technique
Compute V*, Q*, π*	Q-learning
Evaluate a fixed policy π	Value Learning

Model-Free Learning

- act according to current optimal (based on Q-Values)
- but also explore…



Q-Learning

- Q-Learning: sample-based Q-value iteration

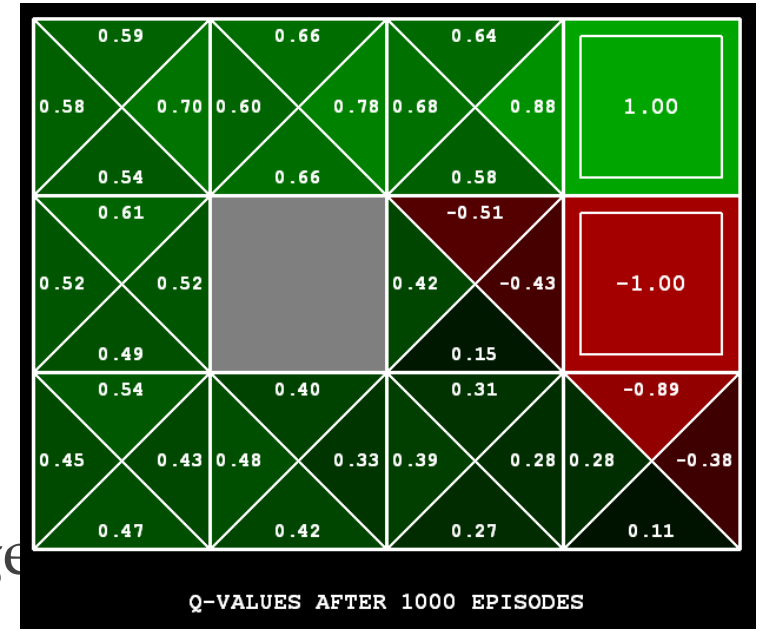
$$Q_{k+1}(s,a) \leftarrow \sum_{s'} T(s,a,s') \left[R(s,a,s') + \gamma \max_{a'} Q_k(s',a') \right]$$

- Learn Q(s,a) values as you go
  - Receive a sample (s,a,s',r)
  - Consider your old estimat$Q(s,a)$
  - Consider your new sample estimate:

$$sample = R(s,a,s') + \gamma \max_{a'} Q(s',a')$$ no longer policy evaluation!

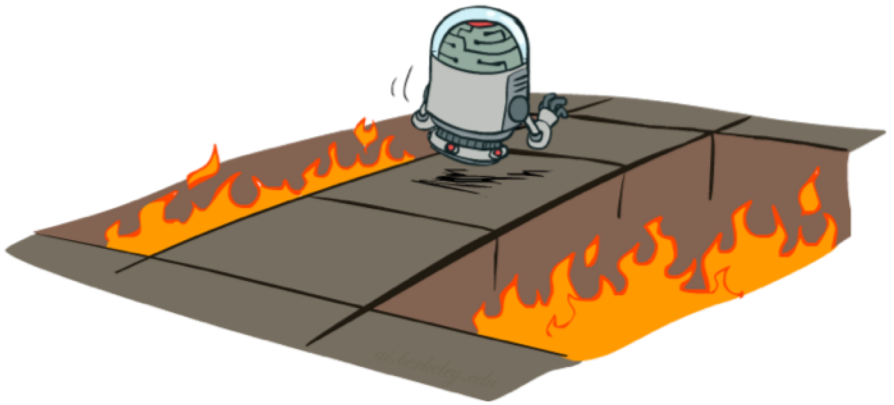
  - Incorporate the new estimate into a running average

$$Q(s,a) \leftarrow (1-\alpha)Q(s,a) + (\alpha)\,[sample]$$



Q-Learning:
act according to current optimal (and also explore…)

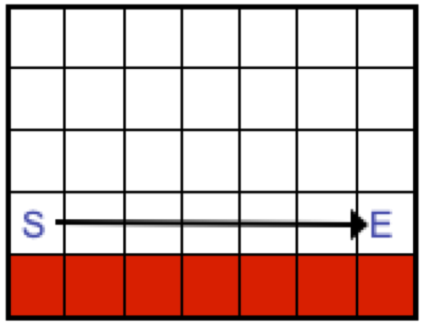
- Full reinforcement learning: optimal policies (like value iteration)
  - You don't know the transitions T(s,a,s')
  - You don't know the rewards R(s,a,s')
  - You choose the actions now
  - Goal: learn the optimal policy / values

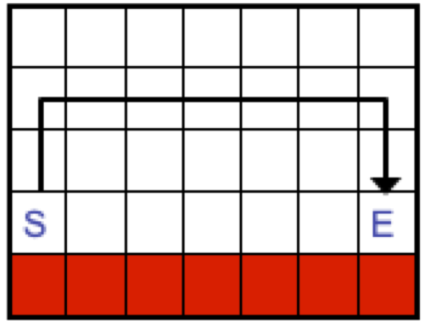


- In this case:
  - Learner makes choices!
  - Fundamental tradeoff: exploration vs. exploitation
  - This is NOT offline planning! You actually take actions in the world and find out what happens…

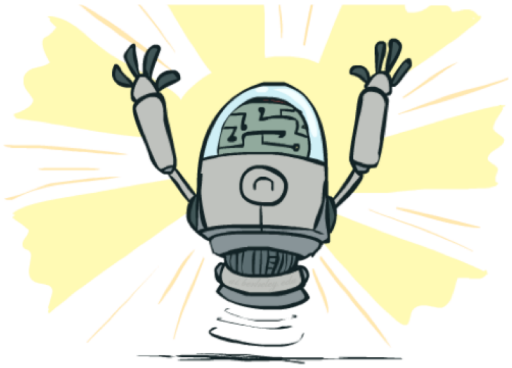
Q-Learning Properties

- Amazing result: Q-learning converges to optimal policy -- even if you're acting suboptimally!

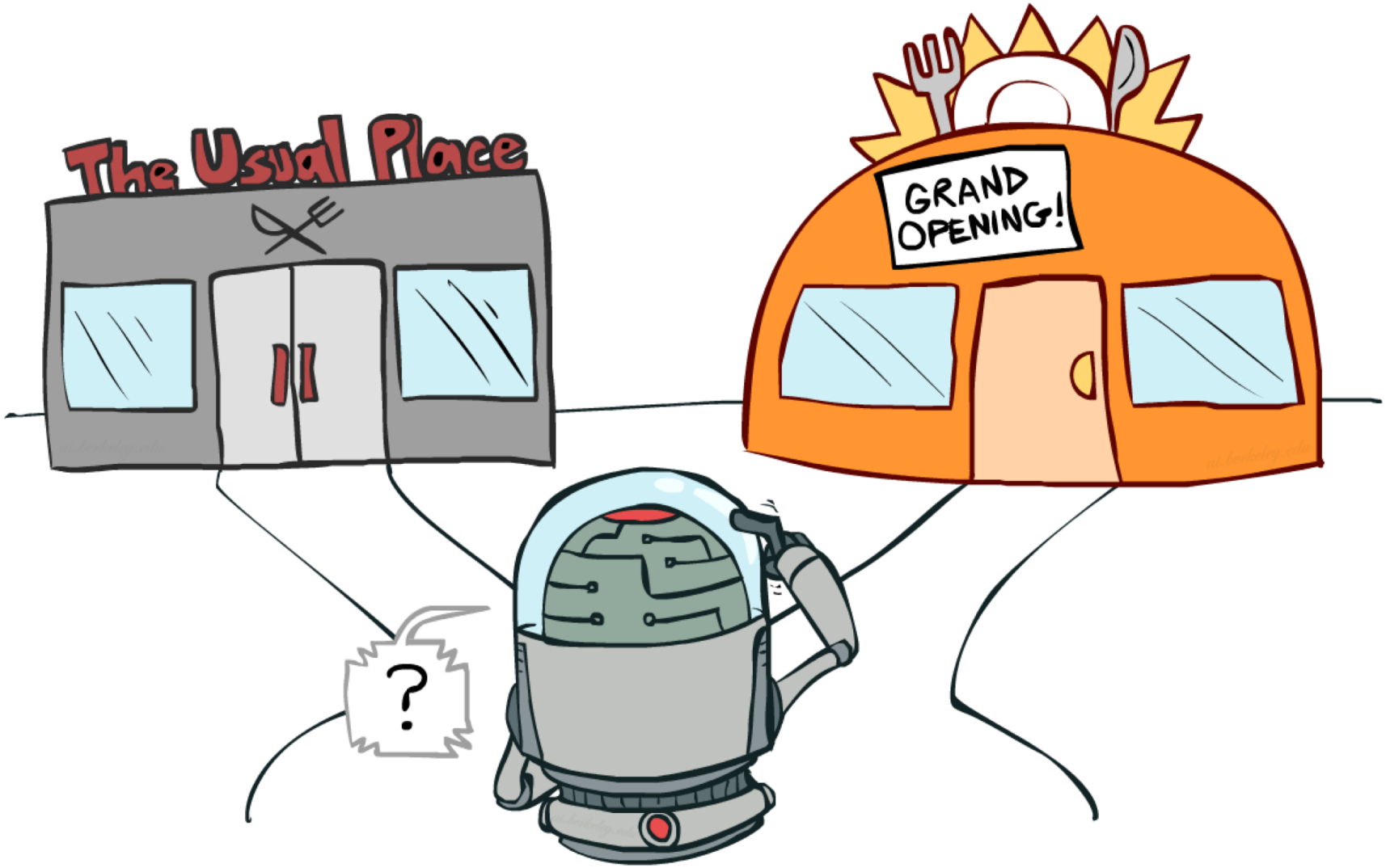




- This is called off-policy learning
- Caveats:
  - You have to explore enough
  - You have to eventually make the learning rate small enough
  - … but not decrease it too quickly
  - Basically, in the limit, it doesn't matter how you select actions



Exploration vs. Exploitation



How to Explore?

- Several schemes for forcing exploration
  - Simplest: random actions (ε-greedy)
    - Every time step, flip a coin
    - With (small) probability ε, act randomly
    - With (large) probability 1-ε, act on current policy
  - Problems with random actions?
    - You do eventually explore the space, but keep thrashing around once learning is done
    - One solution: lower ε over time
    - Another solution: exploration functions

Exploration Functions

- When to explore?
  - Random actions: explore a fixed amount
  - Better idea: explore areas whose badness is not (yet) established, eventually stop exploring

- Exploration function
  - Takes a value estimate u and a visit count n, and returns an optimistic utility, e.g. $f(u, n) = u + k/n$

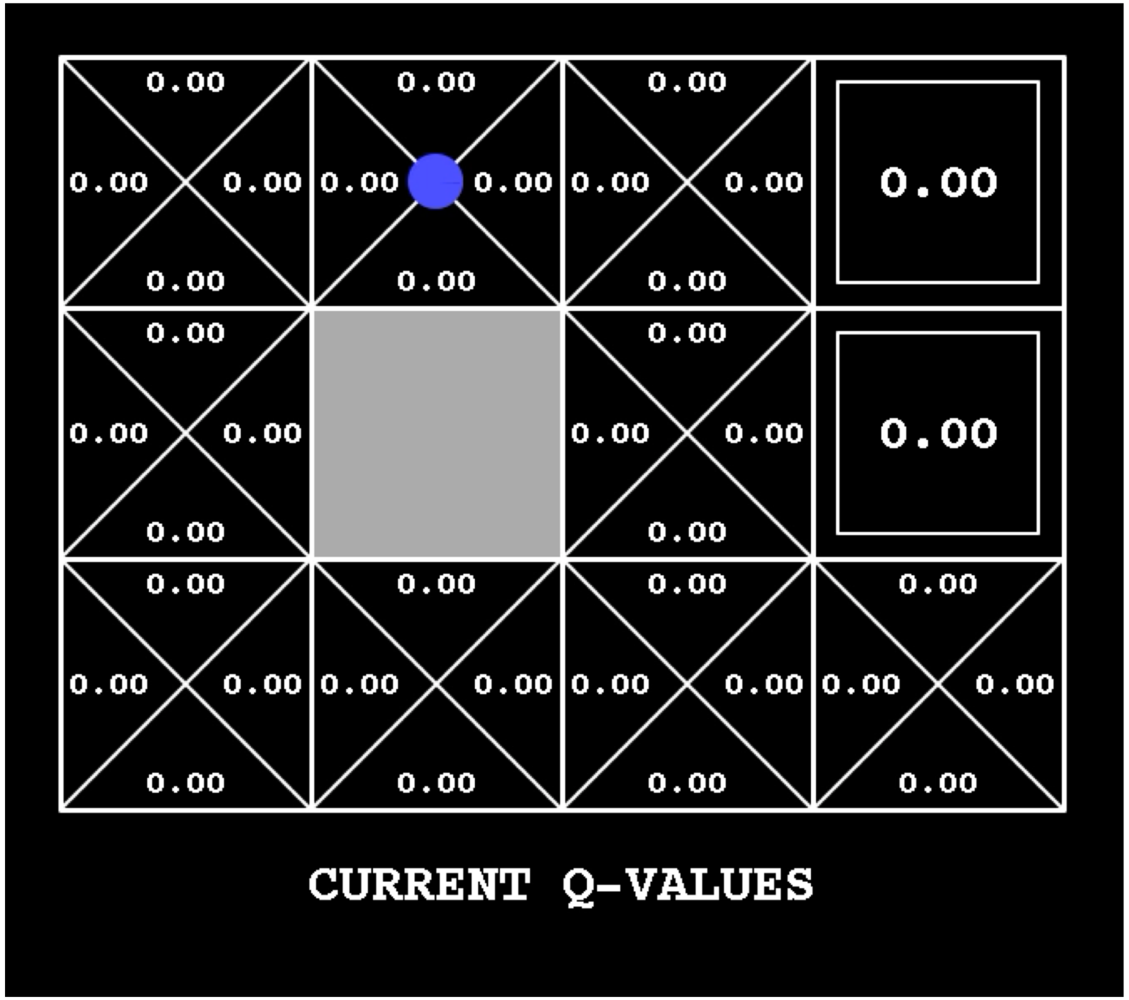
Regular Q-Update: $Q(s,a) \leftarrow_\alpha R(s,a,s') + \gamma \max_{a'} Q(s',a')$

Modified Q-Update: $Q(s,a) \leftarrow_\alpha R(s,a,s') + \gamma \max_{a'} f(Q(s',a'), N(s',a'))$

  - Note: this propagates the "bonus" back to states that lead to unknown states as well!

[Demo: exploration – Q-learning – crawler – exploration function (L11D4)]

Q-Learn Epsilon Greedy



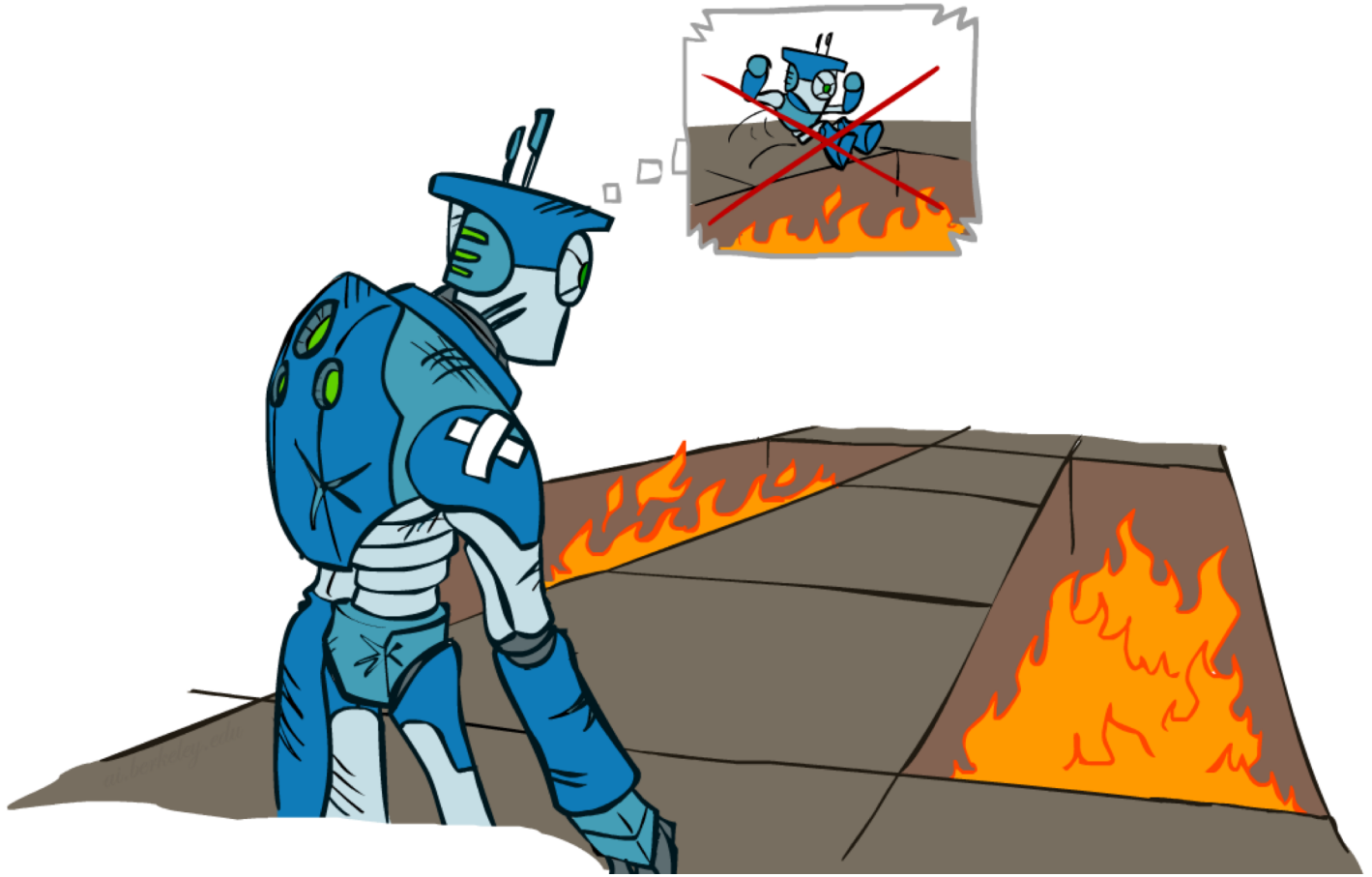
Video of Demo Q-learning – Manual Exploration – Bridge Grid

Video of Demo Q-learning – Epsilon-Greedy – Crawler

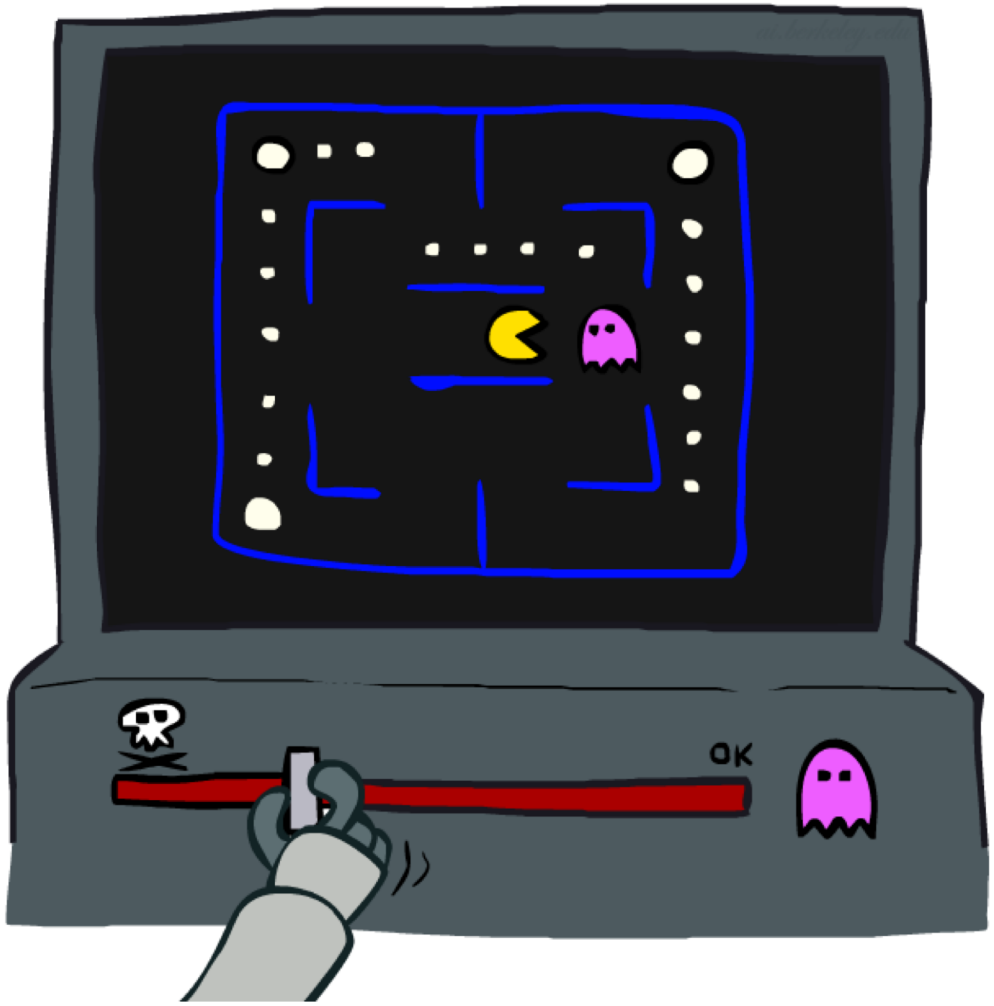
Video of Demo Q-learning – Exploration Function – Crawler

Regret

- Even if you learn the optimal policy, you still make mistakes along the way!
- Regret is a measure of your total mistake cost: the difference between your (expected) rewards, including youthful suboptimality, and optimal (expected) rewards
- Minimizing regret goes beyond learning to be optimal – it requires optimally learning to be optimal
- Example: random exploration and exploration functions both end up optimal, but random exploration has higher regret

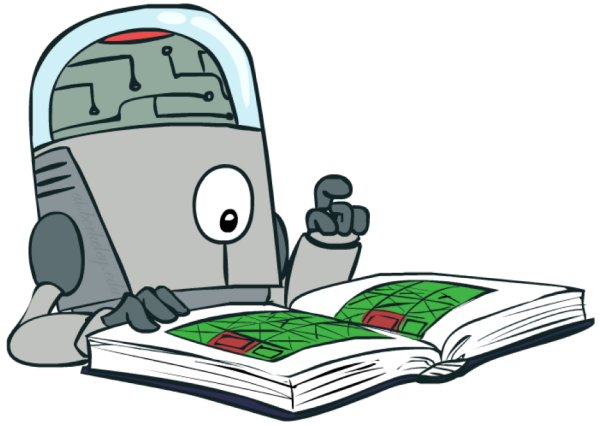


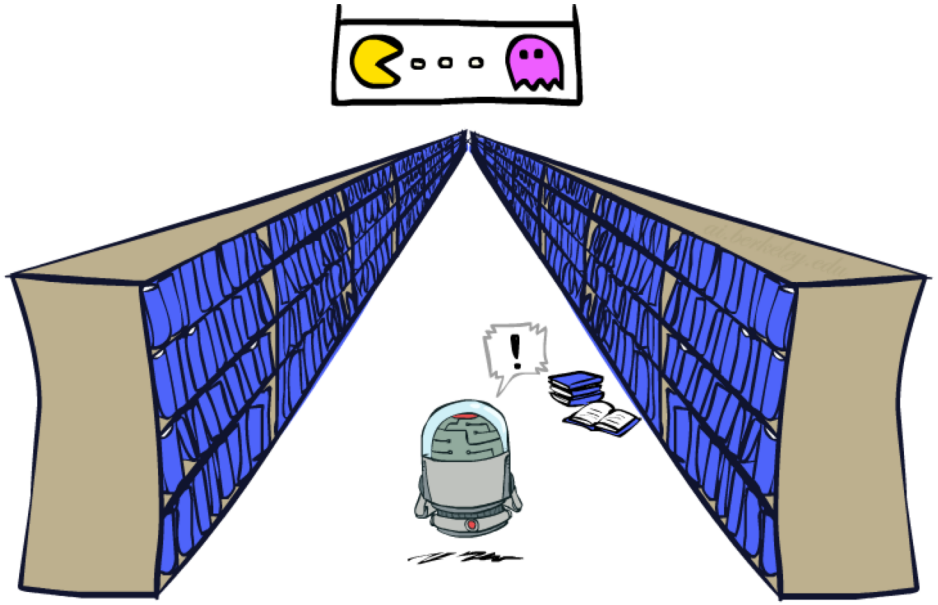
Approximate Q-Learning



Generalizing Across States

- Basic Q-Learning keeps a table of all q-values
- In realistic situations, we cannot possibly learn about every single state!
  - Too many states to visit them all in training
  - Too many states to hold the q-tables in memory
- Instead, we want to generalize:
  - Learn about some small number of training states from experience
  - Generalize that experience to new, similar situations
  - This is a fundamental idea in machine learning, and we'll see it over and over again





[demo – RL pacman]

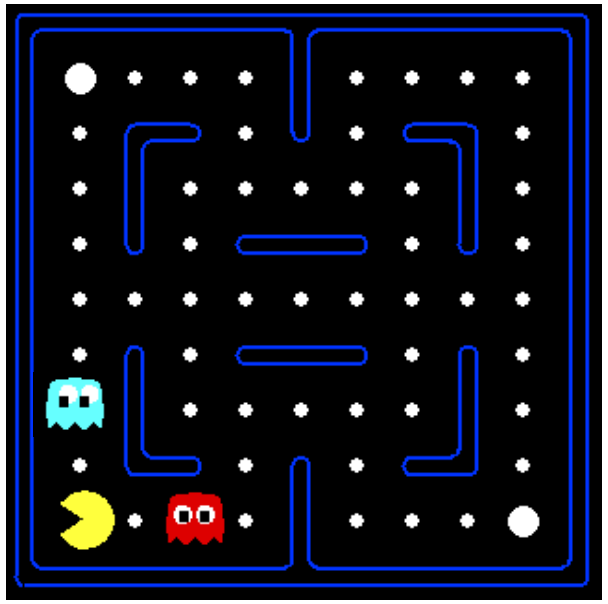
Video of Demo Q-Learning Pacman – Tiny – Watch All

Video of Demo Q-Learning Pacman – Tiny – Silent Train

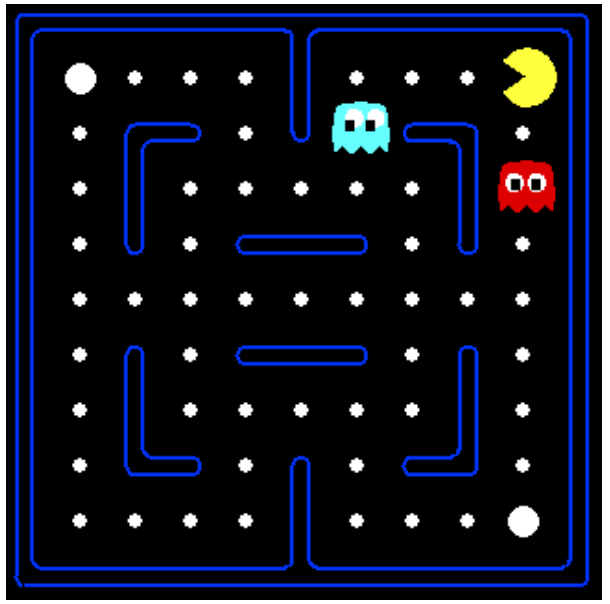
Video of Demo Q-Learning Pacman – Tricky – Watch All

Example: Pacman

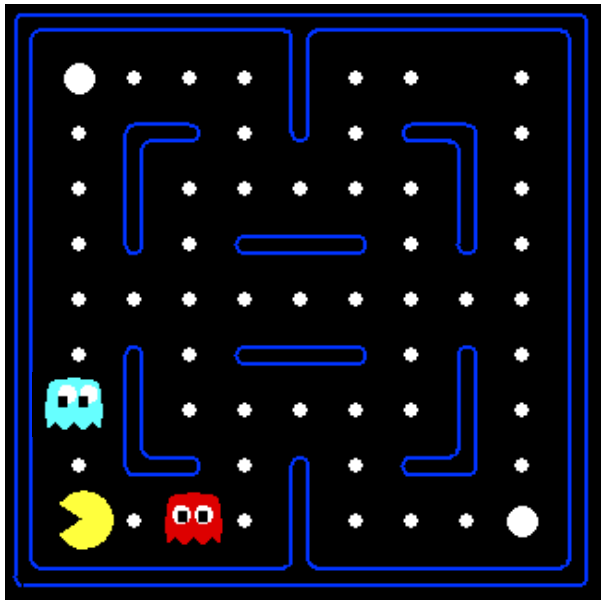
Let's say we discover through experience that this state is bad:



In naïve q-learning, we know nothing about this state:

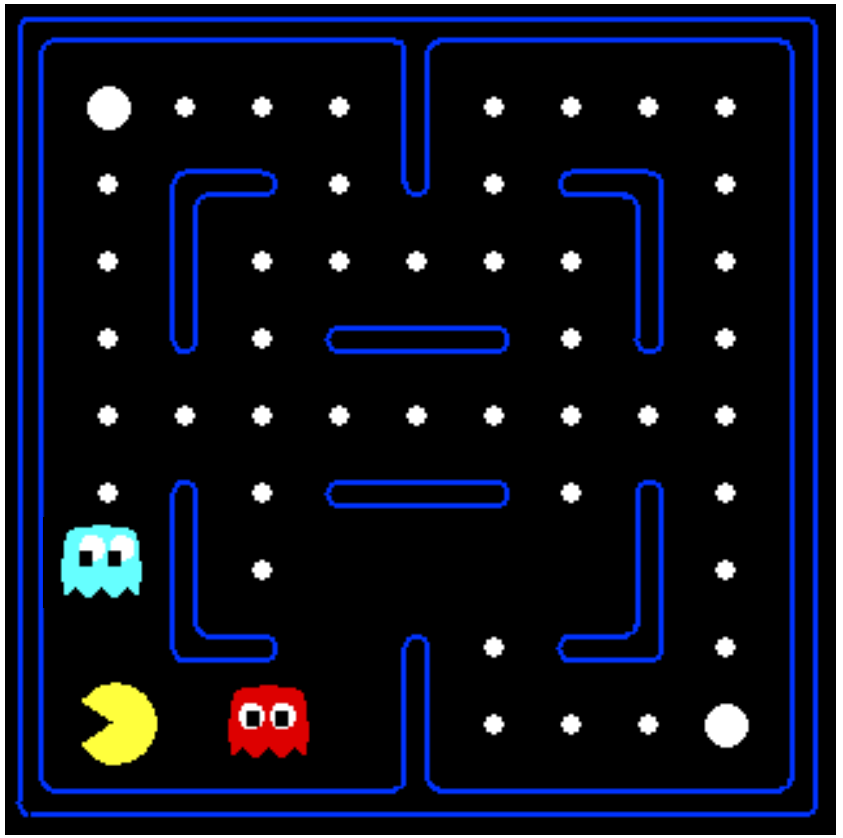


Or even this one!



Feature-Based Representations

- Solution: describe a state using a vector of features (properties)
  - Features are functions from states to real numbers (often 0/1) that capture important properties of the state
  - Example features:
    - Distance to closest ghost
    - Distance to closest dot
    - Number of ghosts
    - 1 / (dist to dot)^2
    - Is Pacman in a tunnel? (0/1)
    - …… etc.
    - Is it the exact state on this slide?
  - Can also describe a q-state (s, a) with features (e.g. action moves closer to food)



Linear Value Functions

- Using a feature representation, we can write a q function (or value function) for any state using a few weights:

$$V(s) = w_1 f_1(s) + w_2 f_2(s) + \ldots + w_n f_n(s)$$

$$Q(s,a) = w_1 f_1(s,a) + w_2 f_2(s,a) + \ldots + w_n f_n(s,a)$$

- Advantage: our experience is summed up in a few powerful numbers

- Disadvantage: states may share features but actually be very different in value!

Approximate Q-Learning

$$Q(s,a) = w_1 f_1(s,a) + w_2 f_2(s,a) + \ldots + w_n f_n(s,a)$$

- Q-learning with linear Q-functions:

$$\text{transition} = (s, a, r, s')$$

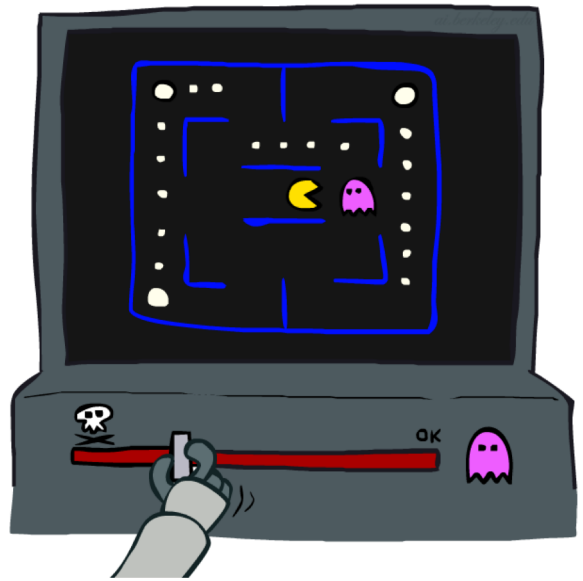
$$\text{difference} = \left[r + \gamma \max_{a'} Q(s', a')\right] - Q(s,a)$$

Exact Q's

$$Q(s,a) \leftarrow Q(s,a) + \alpha\,[\text{difference}]$$

Approximate Q's

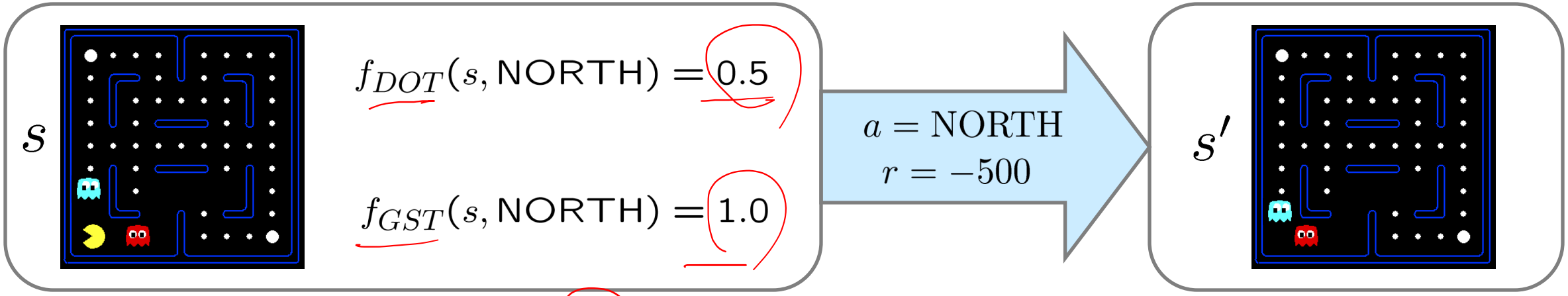
$$w_i \leftarrow w_i + \alpha\,[\text{difference}]\, f_i(s,a)$$



- Intuitive interpretation:
  - Adjust weights of active features
  - E.g., if something unexpectedly bad happens, blame the features that were on: disprefer all states with that state's features
- Formal justification: online least squares

Example: Q-Pacman

$$Q(s,a) = 4.0 f_{DOT}(s,a) - 1.0 f_{GST}(s,a)$$



$Q(s, \text{NORTH}) = +1$

$Q(s', \cdot) = 0$

$$r + \gamma \max_{a'} Q(s', a') = -500 + 0$$

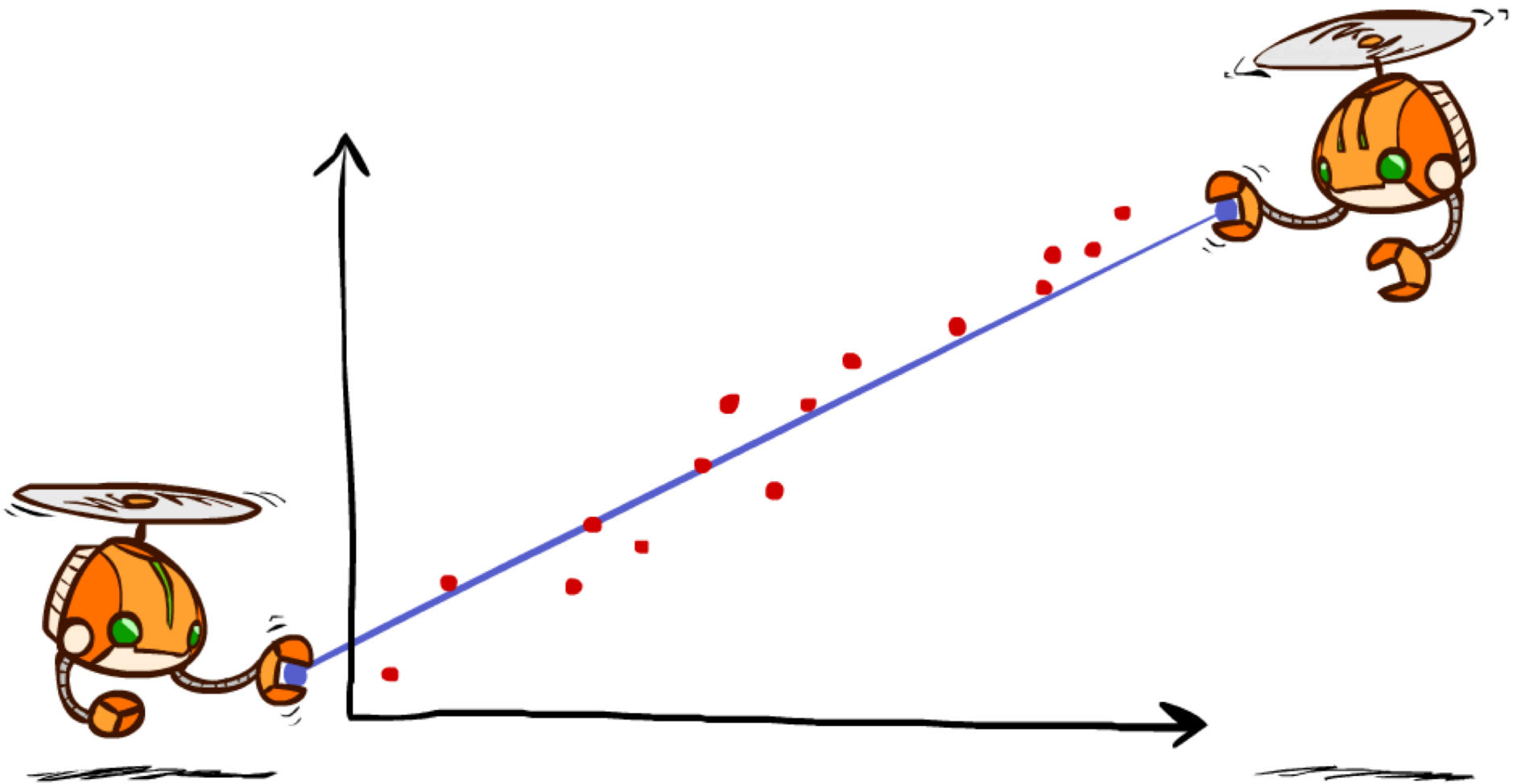
$\text{difference} = -501$

$$w_{DOT} \leftarrow 4.0 + \alpha\,[-501]\,0.5$$
$$w_{GST} \leftarrow -1.0 + \alpha\,[-501]\,1.0$$

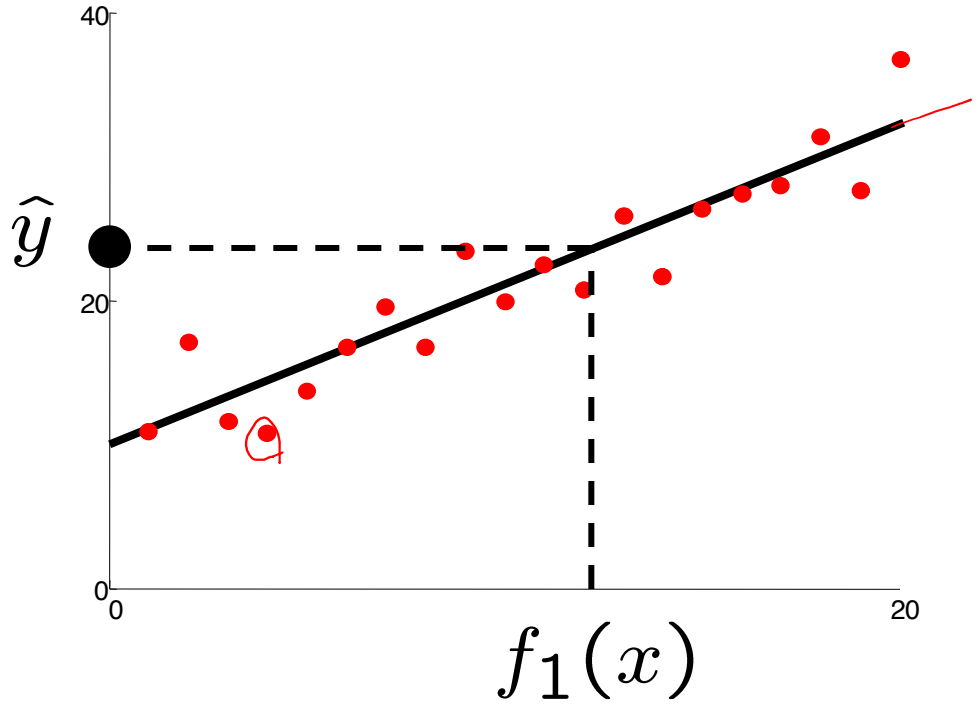
$$Q(s,a) = 3.0 f_{DOT}(s,a) - 3.0 f_{GST}(s,a)$$

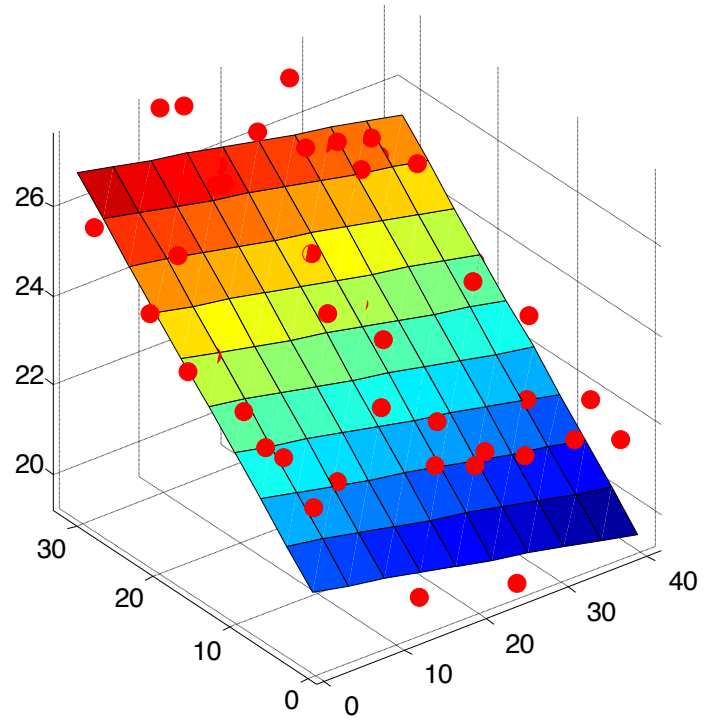
Video of Demo Approximate Q-Learning -- Pacman

Q-Learning and Least Squares



Linear Approximation: Regression





Prediction:

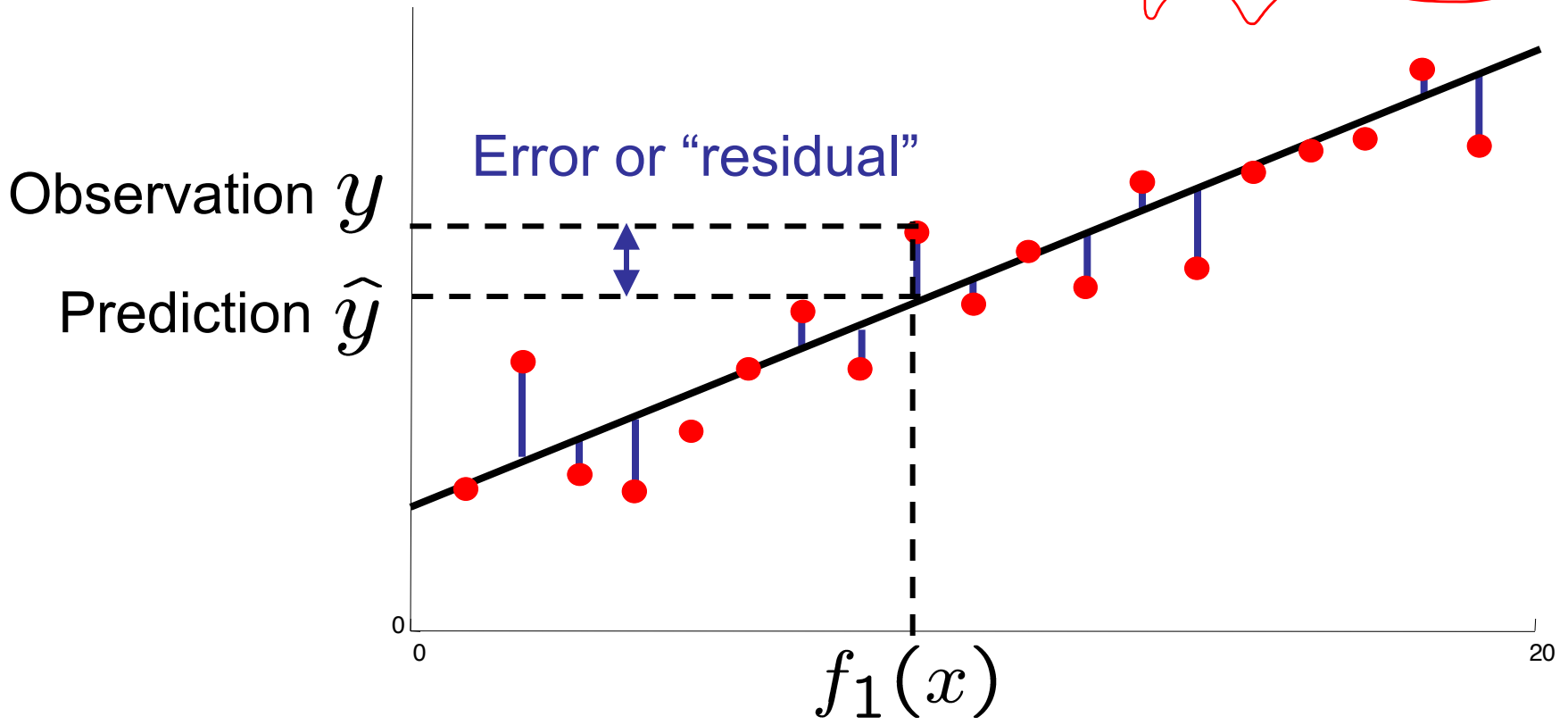
$$\hat{y} = w_0 + w_1 f_1(x)$$

Prediction:

$$\hat{y}_i = w_0 + w_1 f_1(x) + w_2 f_2(x)$$

Optimization: Least Squares

$$\text{total error} = \sum_i (y_i - \hat{y}_i)^2 = \sum_i \left(y_i - \sum_k w_k f_k(x_i) \right)^2$$



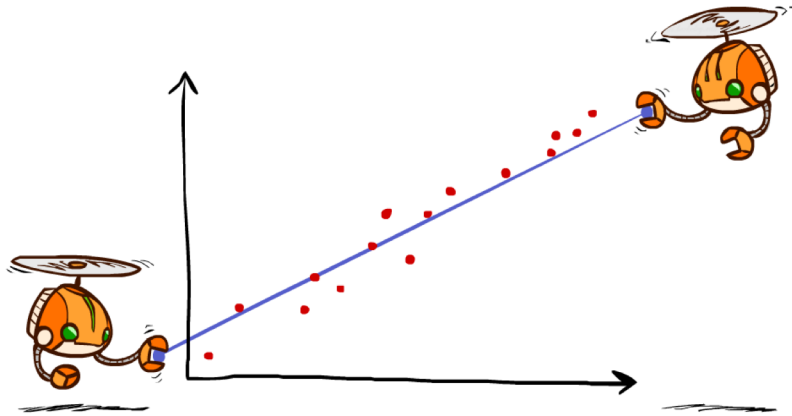
Minimizing Error

Imagine we had only one point x, with features f(x), target value y, and weights w:

$$\text{error}(w) = \frac{1}{2}\left(y - \sum_k w_k f_k(x)\right)^2$$

$$\frac{\partial\ \text{error}(w)}{\partial w_m} = -\left(y - \sum_k w_k f_k(x)\right) f_m(x)$$

$$w_m \leftarrow w_m + \alpha\left(y - \sum_k w_k f_k(x)\right) f_m(x)$$

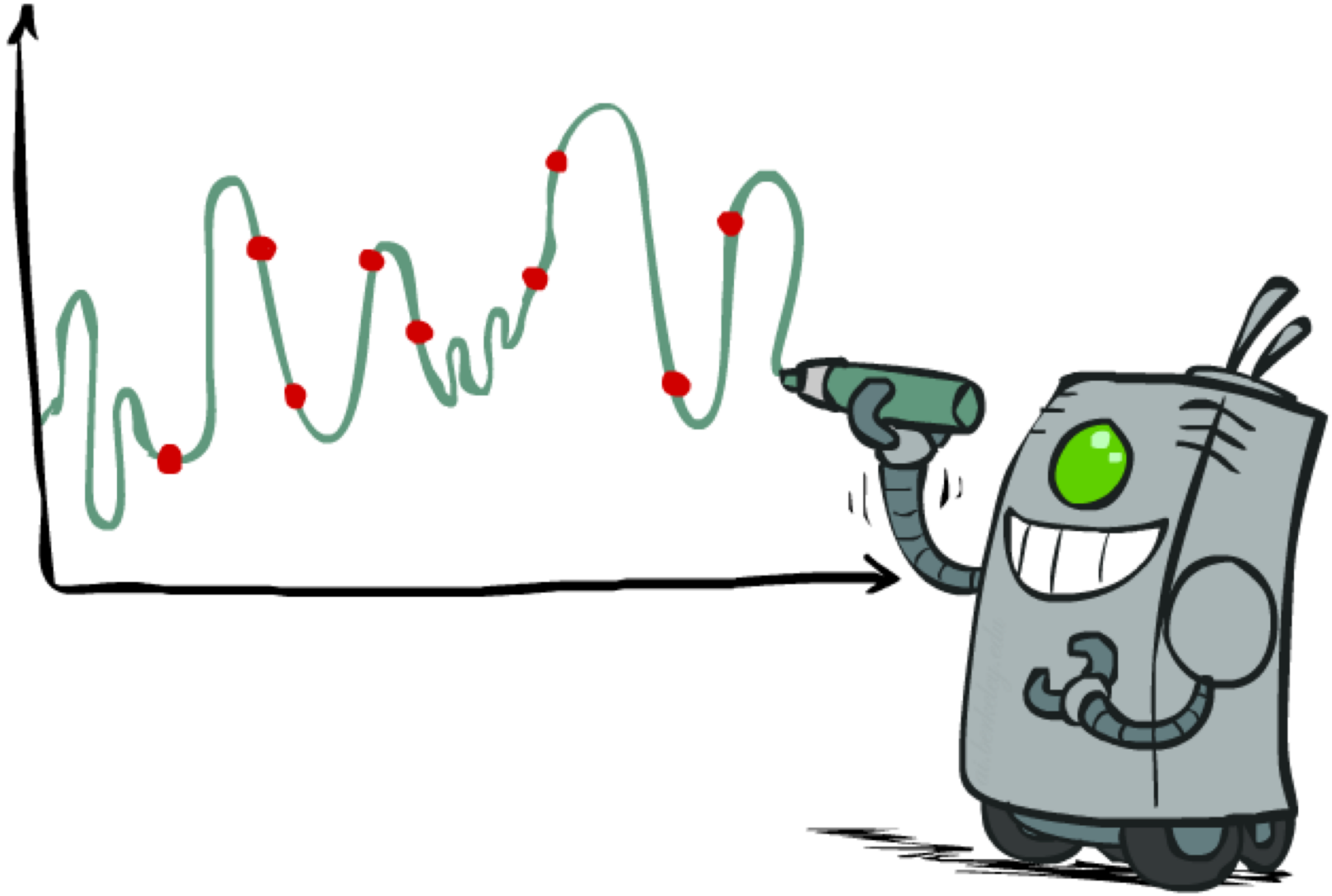


Approximate q update explained:

$$w_m \leftarrow w_m + \alpha\left[r + \gamma \max_a Q(s', a') - Q(s, a)\right] f_m(s, a)$$

"target" "prediction"

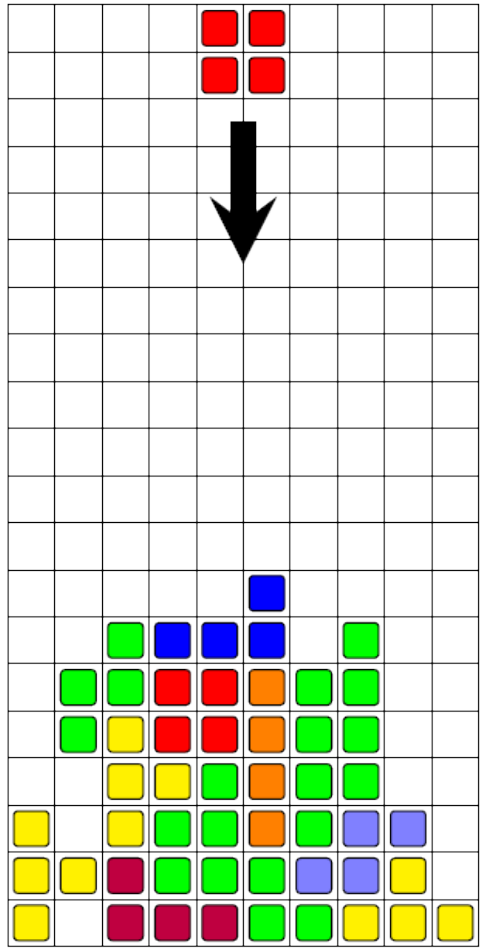
Overfitting: Why Limiting Capacity Can Help



Engineered Approximate Example: Tetris

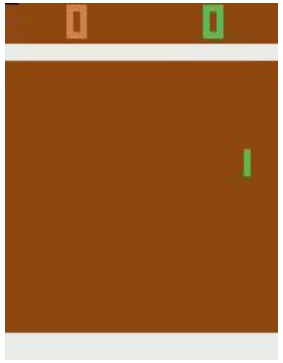
- state: naïve board configuration + shape of the falling piece ~10^{60} states!
- action: rotation and translation applied to the falling piece
- 22 features aka basis functions ϕ_i
  - Ten basis functions, 0, . . . , 9, *mapping the state to the height h[k] of each* column.
  - Nine basis functions, 10, . . . , 18, *each mapping the state to the absolute difference* between heights of successive columns: $|h[k+1] - h[k]|$, $k = 1, \ldots, 9$.
  - One basis function, 19, that maps state to the maximum column height: $\max_k h[k]$
  - One basis function, 20, that maps state to the number of 'holes' in the board.
  - One basis function, 21, that is equal to 1 in every state.

$$\hat{V}_\theta(s) = \sum_{i=0}^{21} \theta_i \phi_i(s) = \theta^\top \phi(s)$$

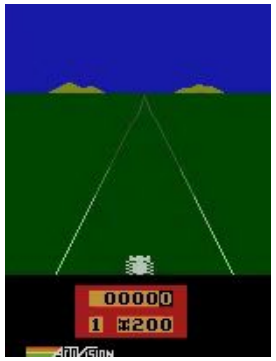


[Bertsekas & Ioffe, 1996 (TD); Bertsekas & Tsitsiklis 1996 (TD); Kakade 2002 (policy gradient); Farias & Van Roy, 2006 (approximate LP)]

Deep Reinforcement Learning

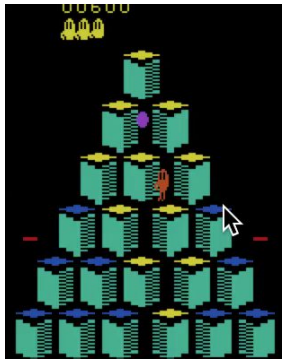


Pong



Enduro

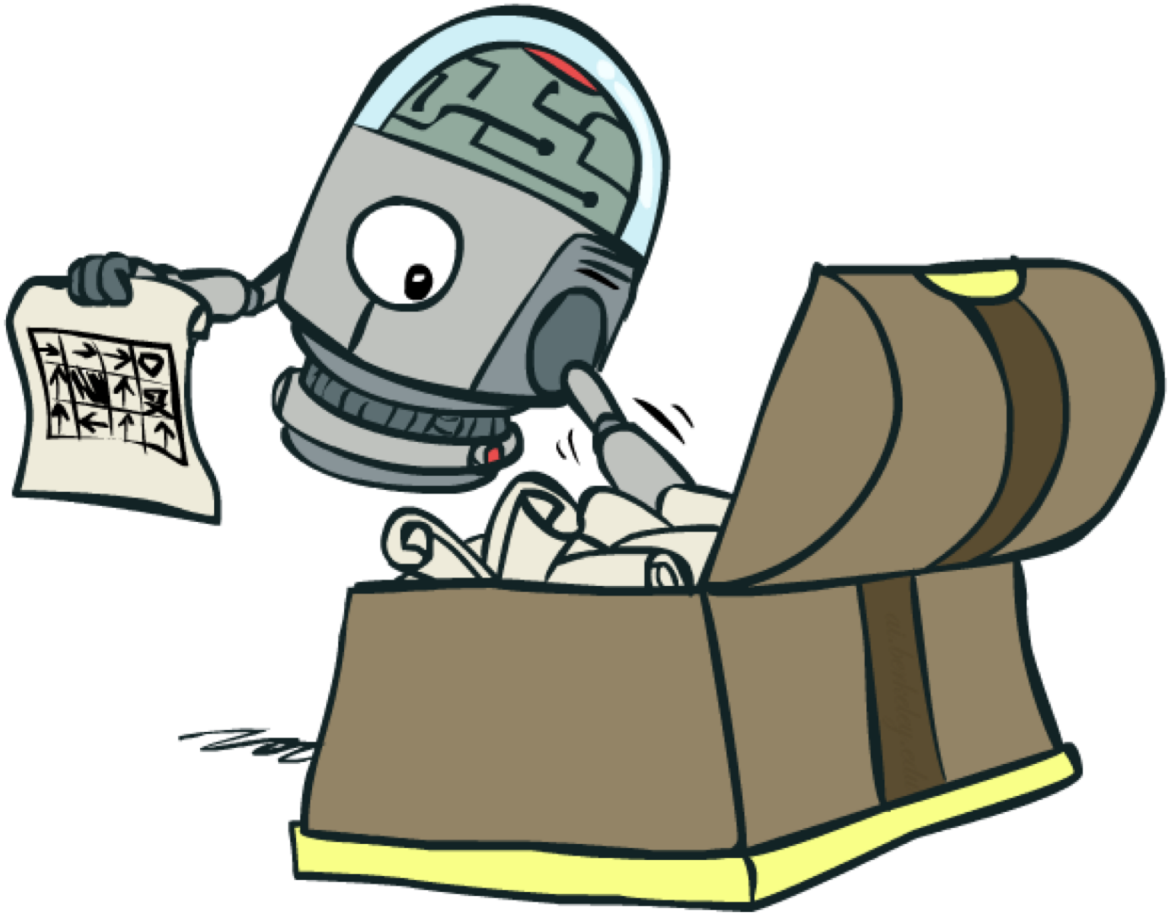
Beamrider



Q*bert

- 49 ATARI 2600 games.
- From pixels to actions.
- The change in score is the reward.
- Same algorithm.
- Same function approximator, w/ 3M free parameters.
- Same hyperparameters.
- Roughly human-level performance on 29 out of 49 games.

Policy Search



Policy Search

- Problem: often the feature-based policies that work well (win games, maximize utilities) aren't the ones that approximate V / Q best
  - E.g. your value functions from project 2 were probably horrible estimates of future rewards, but they still produced good decisions
  - Q-learning's priority: get Q-values close (modeling)
  - Action selection priority: get ordering of Q-values right (prediction)
  - We'll see this distinction between modeling and prediction again later in the course

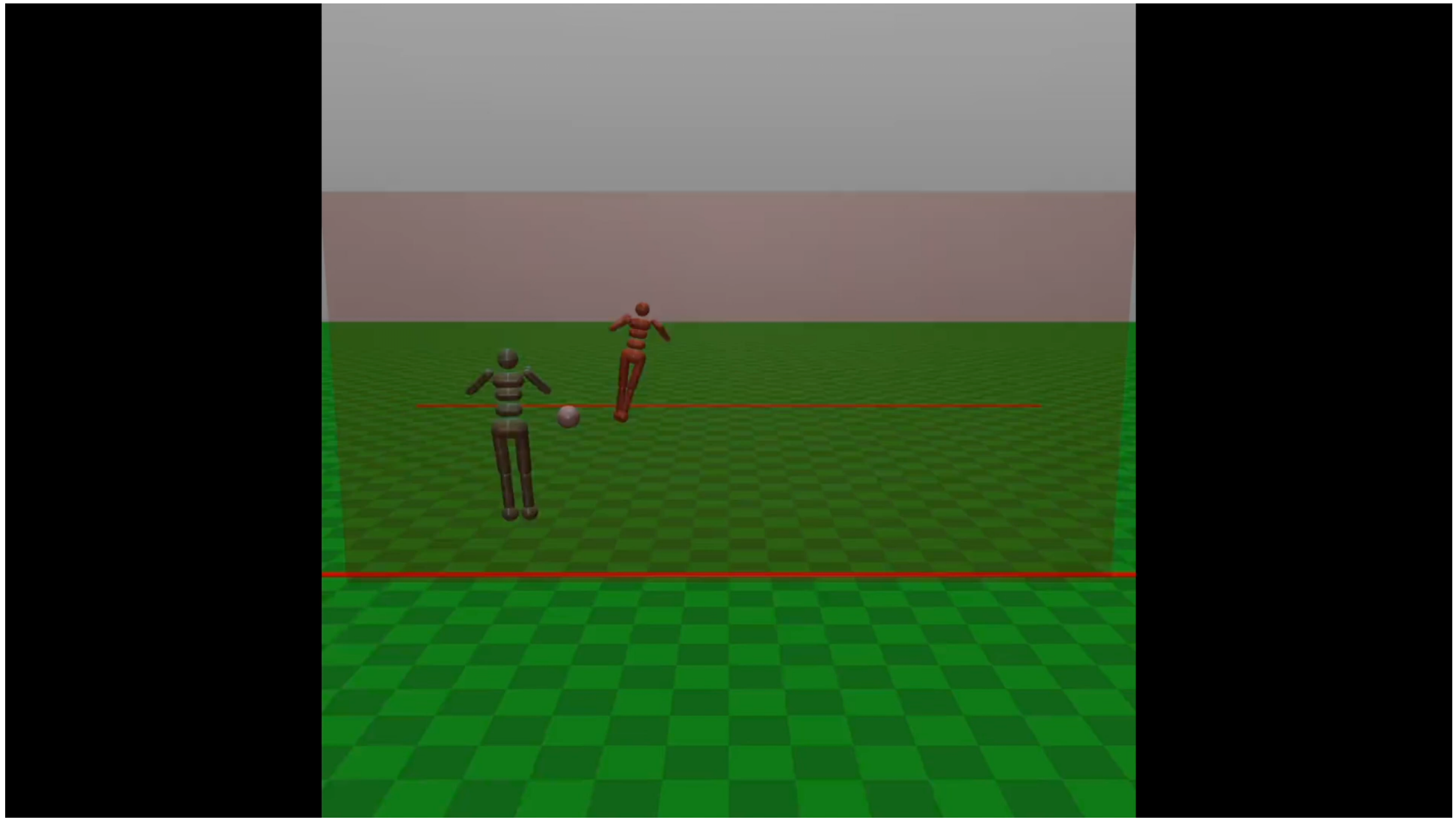
- Solution: learn policies that maximize rewards, not the values that predict them

- Policy search: start with an ok solution (e.g. Q-learning) then fine-tune by hill climbing on feature weights

Policy Search

- Simplest policy search:
  - Start with an initial linear value function or Q-function
  - Nudge each feature weight up and down and see if your policy is better than before
- Problems:
  - How do we tell the policy got better?
  - Need to run many sample episodes!
  - If there are a lot of features, this can be impractical
- Better methods exploit lookahead structure, sample wisely, change multiple parameters…

[Video: GAE]



The Story So Far: MDPs and RL

Known MDP: Offline Solution

Goal	Technique
Compute V*, Q*, π*	Value / policy iteration
Evaluate a fixed policy π	Policy evaluation

Unknown MDP: Model-Based

Goal	*use features to generalize	Technique
Compute V*, Q*, π*		VI/PI on approx. MDP
Evaluate a fixed policy π		PE on approx. MDP

Unknown MDP: Model-Free

Goal	*use features to generalize	Technique
Compute V*, Q*, π*		Q-learning
Evaluate a fixed policy π		Value Learning

Conclusion

- We're done with Part I: Search and Planning!
- We've seen how AI methods can solve problems in:
  - Search
  - Games
  - Markov Decision Problems
  - Reinforcement Learning
- Next up: Uncertainty and Learning!